第1章

汎用フラッシュ・メモリを利用できるコンフィグレーション制御回路を作る

FPGAのコンフィグレーション基礎知識《Altera編》

上島　秀隆

ここでは，米国Altera社のFPGAのうち，主にCyclone Ⅲにおけるコンフィグレーションについて解説する．まず，コンフィグレーションの基本的な方法について説明する．次に，汎用フラッシュ・メモリをコンフィグレーションROMとして用いることのできるコンフィグレーション制御回路を，CPLDを利用して設計する．（編集部）

FPGA（Field Programmable Gate Array）を使用する際に誰もが最初にやるべきことが，コンフィグレーション（Configuration）注1です．

FPGAというデバイスには，なぜコンフィグレーションが必要なのでしょうか．それは，FPGAがSRAMのプロセスを使用して作られているからです．SRAMは電源が入っている時は内容を覚えていますが，電源が切れると内容も消えます．FPGAには回路情報を覚えておくメモリが存在します．コンフィグレーションは，この回路情報を覚えておくメモリにデータを書き込む（回路情報を書き込む）ことをいいます．

米国Altera社のFPGAにおいては，コンフィグレーションに複数の方法があります．本稿では，まず，コンフィグレーションの基本的な方法について説明します．また，リモート・リコンフィグレーションやリモート・アップデートを行えるコンフィグレーション制御回路を設計します．この回路では，コンフィグレーション・データを格納するメモリとして，CFI（Common Flash Interface）を備える汎用フラッシュ・メモリを使用します．

注1：コンフィグレーションは，FPGAの内部で回路情報を記憶するSRAMベースのメモリ・セルにデータを書き込むこと．これに対し，フラッシュ・メモリやEEPROMなどの不揮発性メモリにデータを書き込むことをプログラミングという．

1．コンフィグレーションの基本を理解する

まず，米国Altera社のFPGAでは，コンフィグレーションには，複数の方式あります（表1）．よく使われている順に示すと，次のようになります．

- JTAG（Joint Test Action Group）
- パッシブ・シリアル（PS：Passive Serial）
- アクティブ・シリアル（AS：Active Serial）
- アクティブ・パラレル（AP：Active Parallel）
- ファースト・パッシブ・パラレル（FPP：Fast Passive Parallel）

以下では，それぞれの方式の意図する目的などについて説明し，具体的な方法を示します．

● JTAG

Altera社のFPGAにはJTAGポートがあり，JTAGポートを使ってコンフィグレーションが可能です．コンフィグレーションのための回路を図1に示します．

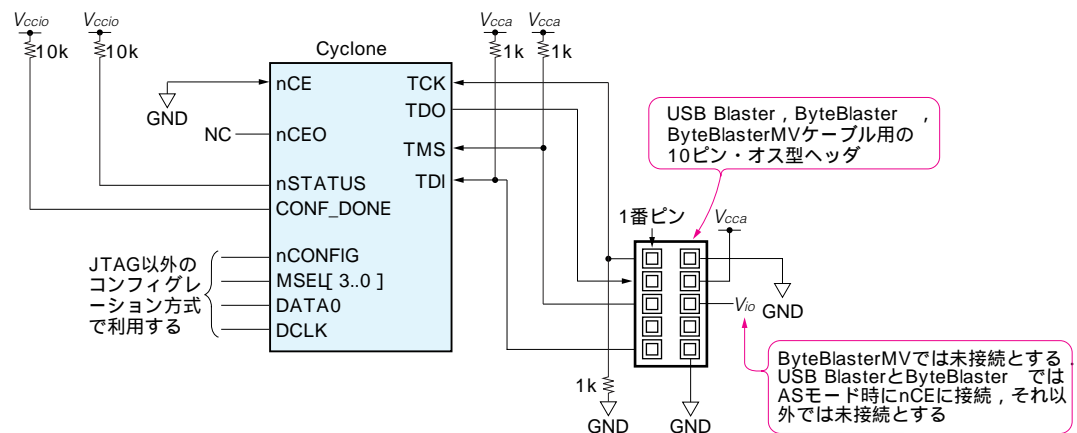
JTAGポートは，Altera社のすべてのFPGAで必ず使えるようにしておいてください．JTAGが使えない基板を設計してしまうと，きっと後悔することになります．

JTAGの本来の用途は，バウンダリ・スキャン・テストです．従って，JTAGポートを備えておけば，基板の

FPGA，Cyclone III，コンフィグレーション，JTAG，パッシブ・シリアル，アクティブ・シリアル，アクティブ・パラレル，ファースト・パッシブ・シリアル，CFIフラッシュ・メモリ，MAX II，Nios II

表1　コンフィグレーション方式

コンフィグレーション・モード	コンフィグレーション方法	復元	リモート・システム・アップグレード	モード設定				コンフィグレーション電圧
				MSEL3	MSEL2	MSEL1	MSEL0	
アクティブ・シリアル・ファースト（AS ファーストPOR）	シリアル・コンフィグレーションROM	○	○	1	1	0	1	3.3V
アクティブ・シリアル・スタンダード（AS スタンダードPOR）	シリアル・コンフィグレーションROM	○	○	0	0	1	0	3.3V
アクティブ・パラレル×16ファースト（AP ファーストPOR）	フラッシュ・メモリ（Intel社の特定品種）	—	○	0	1	0	1	3.3V
		—	○	0	1	1	0	1.8V
アクティブ・パラレル×16（AP スタンダードPOR）	フラッシュ・メモリ（Intel社の特定品種）	—	○	0	1	1	1	3.3V
		—	○	1	0	1	1	3.0V/2.5V
		—	○	1	0	0	0	1.8V
パッシブ・シリアル・ファースト（PS ファーストPOR）	MAX IIやマイクロプロセッサとフラッシュ・メモリ	○	—	1	1	0	0	3.3V/2.5V
	ダウンロード・ケーブル	○						
パッシブ・シリアル・スタンダード（PS スタンダードPOR）	MAX IIやマイクロプロセッサとフラッシュ・メモリ	○	—	0	0	0	0	3.3V/2.5V
	ダウンロード・ケーブル	○						
ファースト・パッシブ・パラレル・ファースト（FPP ファーストPOR）	MAX IIやマイクロプロセッサとフラッシュ・メモリ	—	—	1	1	1	0	3.3V/2.5V
		—	—	1	1	1	1	1.8V/1.5V
JTAGベース	MAX IIやマイクロプロセッサとフラッシュ・メモリ	—	—	JTAG以外で使用するモードに合わせる				—
	ダウンロード・ケーブル	—	—					

図1
JTAG方式の接続方法
JTAGポートは，すべてのFPGAで使えるようにしておくことを推奨．

チェックに使用できます．デバッグ時に，米国Tektronix社や米国Agilent Technologies社のロジック・アナライザを使用する場合でも，ロジック・アナライザ・インターフェース（LAI）としてJTAGを使用します．Altera社のオンチップ・ロジック・アナライザ機能であるSignalTap IIを使う際にも必要です．このほか，ソフト・マクロのCPUコアであるNios IIを活用する場合も利用します．

ときどき，JTAGインターフェースのないFPGAボードを目にすることがありますが，デバッグで困らないのかと心配になります．コネクタの実装スペースがない場合などでは，変換ケーブルを使うようにしても構わないので，JTAGポートは必ず使えるようにしておきましょう．

● パッシブ・シリアル

パッシブ・シリアル方式は，JTAGポートを使用しない（できない）時に，ホスト・パソコンから，FPGAを直接コンフィグレーションする場合に使用するモードです．また，CycloneとCyclone II，Cyclone IIIのピン数の少ないパッケージの品種（表2）でリモート・リコンフィグレーションやリモート・アップデートを行う場合や，外部のCPUや

CPLD (Complex Programmable Logic Device) を使ってコンフィグレーションする場合に使用します．コンフィグレーションのための回路を図2に示します．

Cyclone IIIのうち324ピン以上のパッケージの品種では，この方法を使わなくても，アクティブ・パラレルやファースト・パッシブ・パラレルがサポートされているので，そちらのモードの利用をお勧めします．

● アクティブ・シリアル

アクティブ・シリアル方式は，専用シリアル・コンフィグレーションROMを使用する際の最も基本的なモードです．

接続が一番簡単で，ユーザI/Oピンも消費しません．できるだけ多くのI/Oピンを使用したい（コンフィグレーションに必要なピン数を減らしたい）場合に，このモードを選択します．実装面積的にも一番有利です．

Altera社のFPGAのうち，CycloneシリーズとStratixシリーズ，Arria GXで使用できます．

使用可能なコンフィグレーションROMは，Altera社のEPCS1/4/16/64/128です．

表2　Cyclone IIIでサポートされるコンフィグレーション方式

型　名	パッケージ					
	E144	Q240/U256	F256/U484	F324	F484/U484	F780
EP3C5	AS, PS, JTAG	―	AS, PS, FPP, JTAG	―	―	―
EP3C10		―		―	―	―
EP3C16		AS, PS, FPP, JTAG		―	○	―
EP3C25				○	―	―
EP3C40	―		―	○	○	○
EP3C55	―	―	―	―	○	○
EP3C80	―	―	―	―	○	○
EP3C120	―	―	―	―	○	○

（E144～F256/U484）アクティブ・シリアルはサポートされていない

（F324～F780）すべての方式（AS，AP，PS，FPP，JTAG）がサポートされている

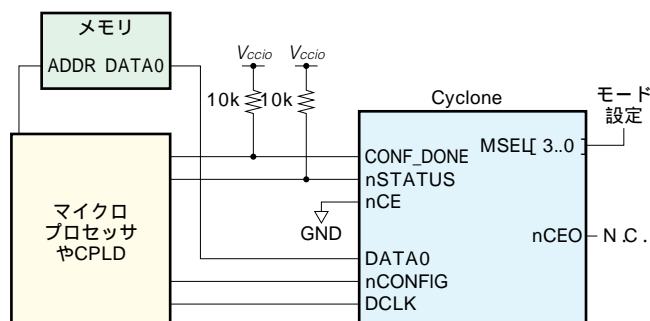



図2　パッシブ・シリアル方式の接続方法

Cyclone IIIにおけるアクティブ・シリアル方式のコンフィグレーション回路を図3に示します．

Cyclone/Cyclone II/Cyclone IIIでは，専用シリアル・コンフィグレーションROMへJTAGを使ってプログラム可能です．従って，アクティブ・シリアル用のヘッダ・ピンを用意しなくても構いません．

このモードでは，リモート・リコンフィグレーションやリモート・アップデートは使えません．

● アクティブ・パラレル

アクティブ・パラレル方式は，Cyclone IIIのうち324ピン以上のパッケージの品種でのみ可能になった新しいモードです．

汎用CFIフラッシュ・メモリ（Intel社のP30/P33を推奨）を使用して，リモート・リコンフィグレーションやリモート・アップデートを実現できます．

アクティブ・パラレル方式によるコンフィグレーション回路を図4に示します．

● ファースト・パッシブ・パラレル

ファースト・パッシブ・パラレル方式は，StratixやStartix II，Stratix II GX，Arria GX系とCyclone IIIの多ピン・パッケージで可能なモードです．

いずれも外部にコンフィグレーション制御回路が必要になります．このモードでもリモート・リコンフィグレーションやリモート・アップデートを実現できます．

ファースト・パッシブ・パラレル方式によるコンフィグレーション回路を図5に示します．

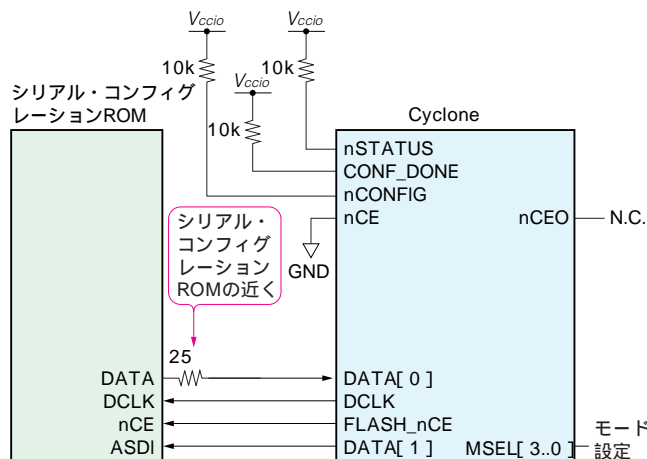



図3　アクティブ・シリアル方式の接続方法

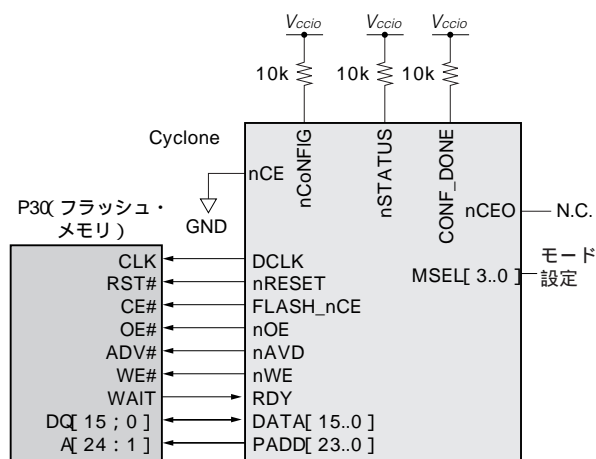
● **複数のFPGAをコンフィグレーションする**

複数のFPGAをコンフィグレーションする場合の回路を**図6**に示します．

シリアルまたはパラレルのコンフィグレーション方式のチェーンと，JTAGのチェーンを一致させてください．すなわち，nCE→nCEO→nCE→…のチェーンとTDI→TDO→TDI→…のチェーンが一致するようにします．

FPGAによってI/O（コンフィグレーション・ピンやJTAG）の電源が変わる場合があるので注意が必要です．また，異なるV_{ccio}電圧のFPGAをチェーンする場合は，V_{oh}/V_{ih}，V_{ol}/V_{il}の関係を考慮して接続してください．

2. CPLDによるコンフィグレーション制御回路の設計

ここでは，CFIフラッシュ・メモリを使用して，複数のコンフィグレーション・データを切り替えられるシステムを設計する方法について説明します．コンフィグレーション制御回路には，CPLD（Altera社のMAX II）を使用します．

● **二つのコンフィグレーション・データを切り替える**

リモート・アップデートを行うシステムでは，アップデート・データに不具合が見つかった場合，初期状態に戻す機能が必要です．そこで，一つはセーフ・モードのコンフィグレーション・データとして保持し続け，もう一方の領域のデータをアップデート・データとして変更するような使い方をします．

そこで，フラッシュ・メモリに二つのコンフィグレーション・データを持たせることにします（**図7**）．CFIフラッシュ・メモリとして，米国Intel社のデータ幅が8ビットまたは16ビットの品種を想定します．

● **FPGAによって使用するモードが変わる**

ターゲットとするFPGAの種類（パッケージにもよる）によって，パッシブ・シリアル方式か，ファースト・パッシブ・パラレル方式のいずれかを選択します．

Cyclone/Cyclone II/Cyclone IIIのピン数が少ないパッケージの品種では，パッシブ・シリアル・モードを使用す

図4　アクティブ・パラレル方式の接続方法

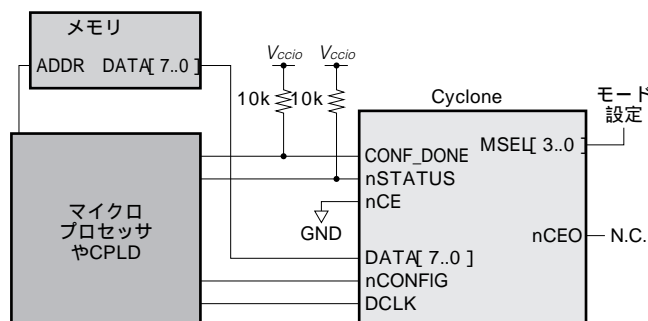



図5　ファースト・パッシブ・パラレルの接続方法

コラム　提供するプロジェクトについて

今回設計したコンフィグレーション制御回路のVerilog HDLソース・コードと，MAX II向けプロジェクトは，本誌Webサイト（`http://www.cqpub.co.jp/dwm/`）からダウンロードできます．

この回路は，あるボード向けの設計が元になっています．その後，いくつかの改定をしました．そのため，テストベンチはどちらかと言うと可読性はあまり良くありません．しかしシミュレーションで抑えるところは一応抑えたつもりです．

コンフィグレーション制御回路のクロックは，MAX II内蔵の発振回路を使用しています．ただし，元になった設計では33MHzの発振器からのクロックをソースにしていました．parameterで定義しているタイミング・パラメータを変更すれば，周波数を変更できます．

る必要があります．Cyclone IIIのうち324ピン以上のパッケージの品種では，アクティブ・パラレル方式とファースト・パッシブ・パラレル方式のいずれも使用できます．Stratix/ Stratix II /Arria GX/Stratix IIIは，ファースト・パッシブ・パラレル方式を使用します．

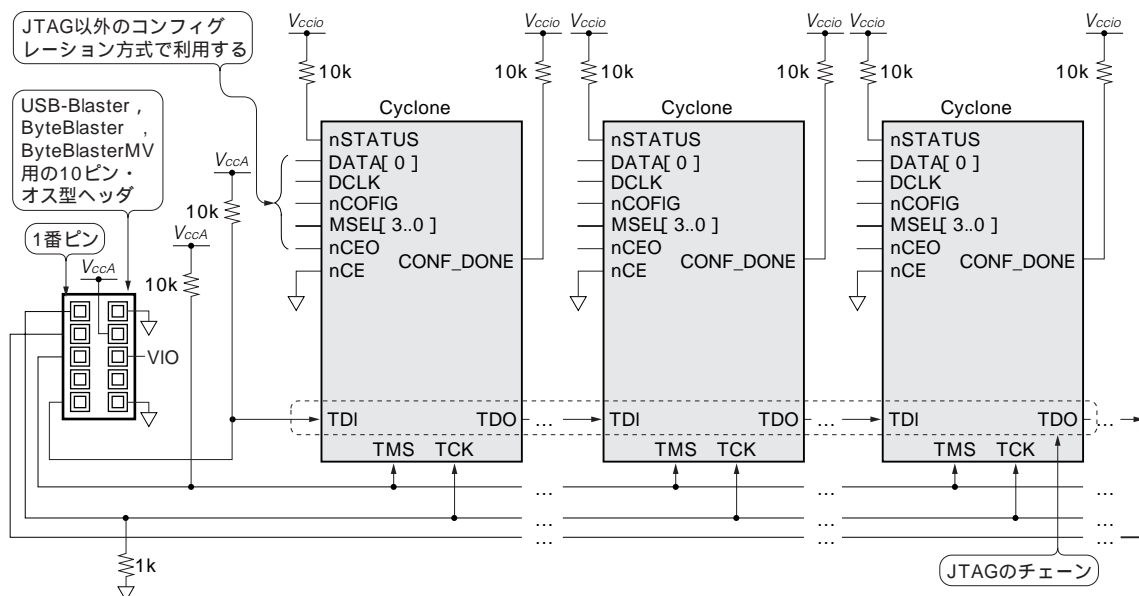
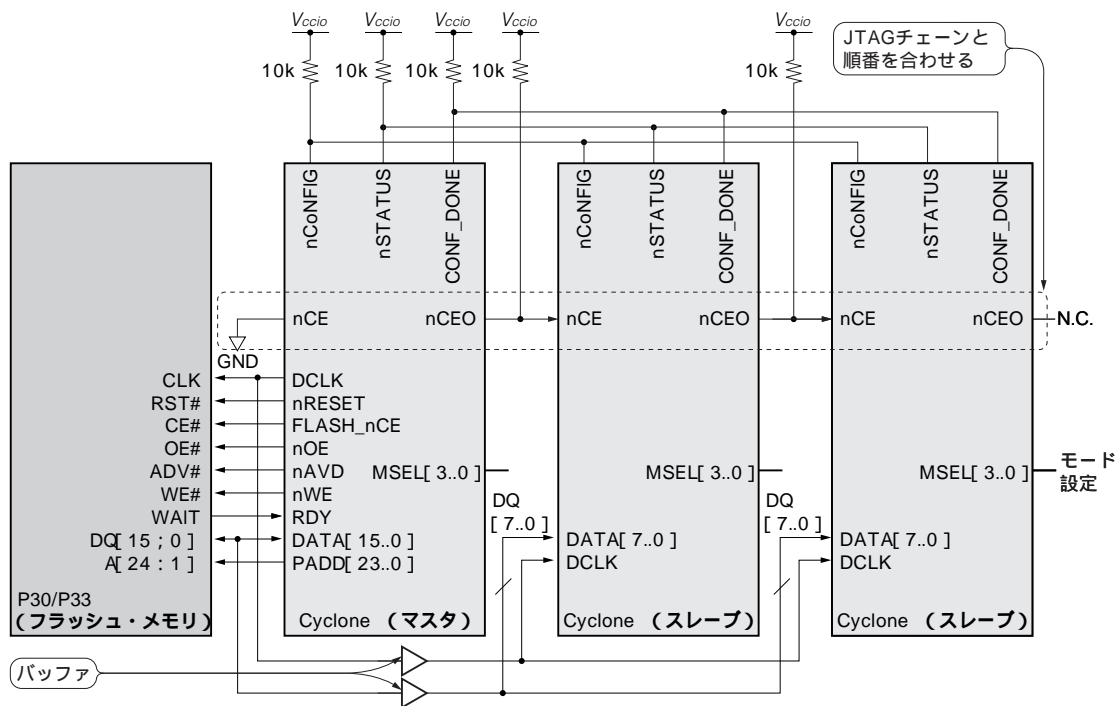
今回はFPGAとしてCyclone II（EP2C35またはEP2C50）を使用するものと想定します．

（a） JTAG方式

（b） アクティブ・パラレル方式

図6　複数のFPGAをコンフィグレーションする接続方法

シリアルまたはパラレルのコンフィグレーション方式のチェーンと，JTAGのチェーンを一致させる．すなわち，nCE→nCEO→nCE→…のチェーンとTDI→TDO→TDI→…のチェーンを合わせる．

● コンフィグレーション制御回路の設計

CPLDを使ったPSモード用コンフィグレーション制御回路の例を**図8**に示します，使用するフラッシュ・メモリの容量により，アドレス・バスの本数が変わります．フラッシュ・メモリは，バイト・モードで使用しているので， DQ15はアドレス線A-1になります．

CPLDに実装する機能のVerilog HDL記述を**リスト1**に示します．

設計したコンフィグレーション制御回路は，コンフィグレーションを行うときにのみフラッシュ・メモリをアクセスします（p.69のコラム「CFIフラッシュ・メモリへのコンフィグレーション・データの書き込み」を参照）．つまり，CPLDがフラッシュ・メモリにアクセスしている時は，FPGAはコンフィグレーション中なので，FPGA側のI/Oピンはすべてトライステート状態になります．

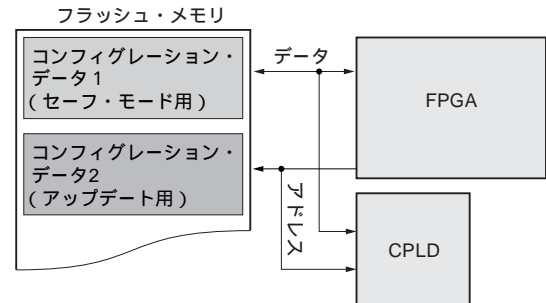



図7　フラッシュ・メモリに二つのコンフィグレーション・データを持たせる

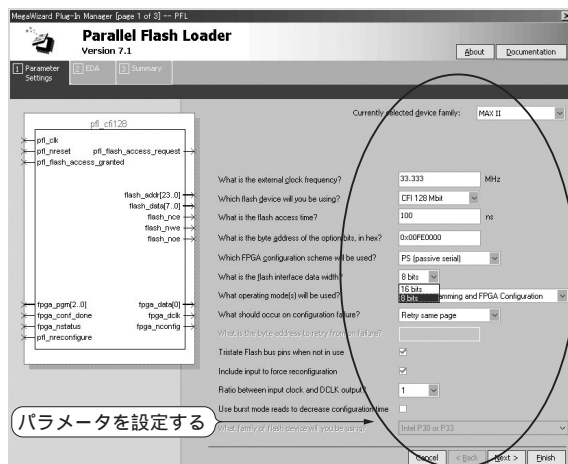
例えばリモート・アップデートを行うシステムでは，アップデート・データに不具合が見つかった場合に初期状態に戻す機能が必要である．そこで，一つはセーフ・モードのコンフィグレーション・データとして保持し続け，もう一方の領域のデータをアップデート・データとして変更するような使い方をする．

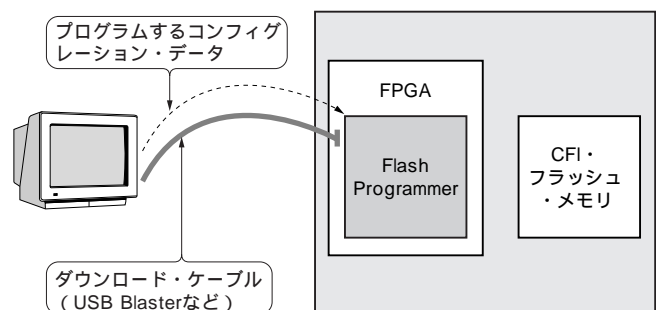
コラム　CFIフラッシュ・メモリへのコンフィグレーション・データの書き込み

パッシブ・シリアル・モードとファースト・パッシブ・パラレル・モードを使用する場合，CFIフラッシュ・メモリへのコンフィグレーション・データの書き込み機能はありません．PFL（Parallel Flash Loader）というマクロを使用するか，Nios II Flash Programmerを使用する必要があります．

PFLは，CPLDによりコンフィグレーション制御回路を構成する際に使用するIPコアです．Quartus IIに標準で付属します．ウィザード形式（MegaWizard Plug-In Manager）で機能を設定するので，ある程度の自由度はありますが，それなりに制限もあります（**図A-1**）．フラッシュ・メモリの書き込みができる点は便利です．PFLをMAX IIに組み込む場合は，EPM570以上の規模が必要になります．コンフィグレーション制御機能だけであればEPM240を利用できます．

Nios II Flash Programmerは，ソフト・マクロのCPU「Nios II」の開発ツールであるNios II IDEからフラッシュ・メモリの書き込みを行うためのツールです（**図A-2**）．従って，この方法は，Nios IIを組み込んだ設計に向いています．

Nios IIを組み込まないのであれば，PFLを使用する方が無難です．Nios IIを組み込む場合はNios II Flash Programmerを使い，Nios IIを使わない場合はPFLを使うことをお勧めします．

図A-1　PFLのウィザード画面

CPLDによりコンフィグレーション制御回路を構成する際に使用するIPコアである．

図A-2　Nios II Flash Programmer

ソフト・マクロのCPU「Nios II」の開発ツールであるNios II IDEからフラッシュ・メモリの書き込みを行うためのツールを使って，フラッシュ・メモリにデータを書き込む．

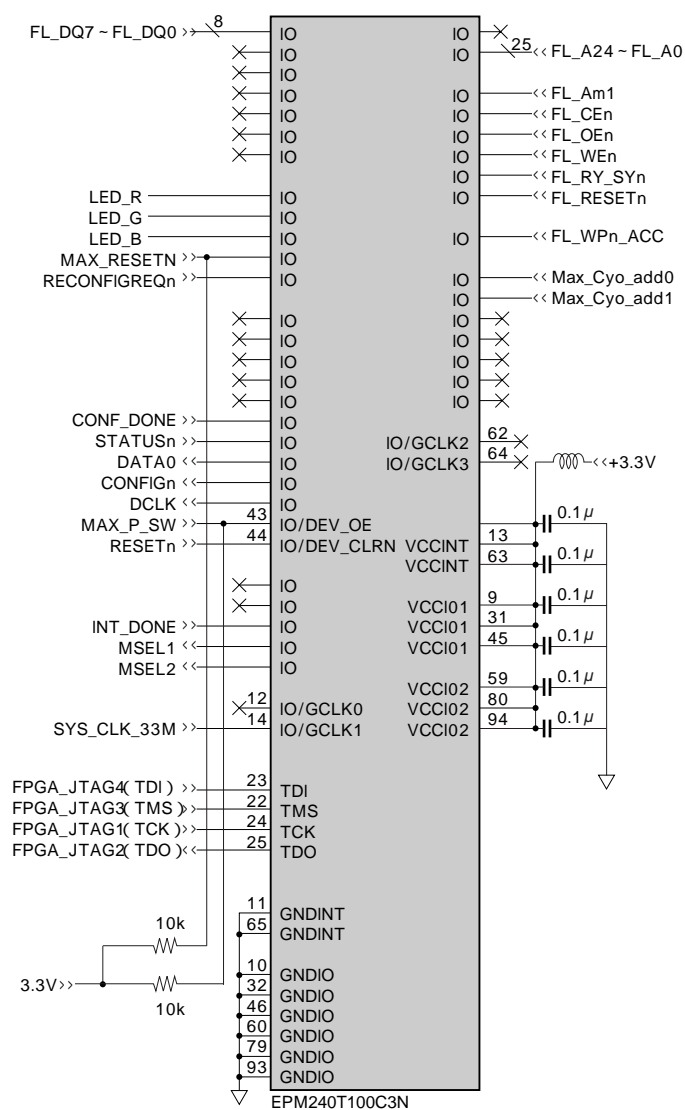
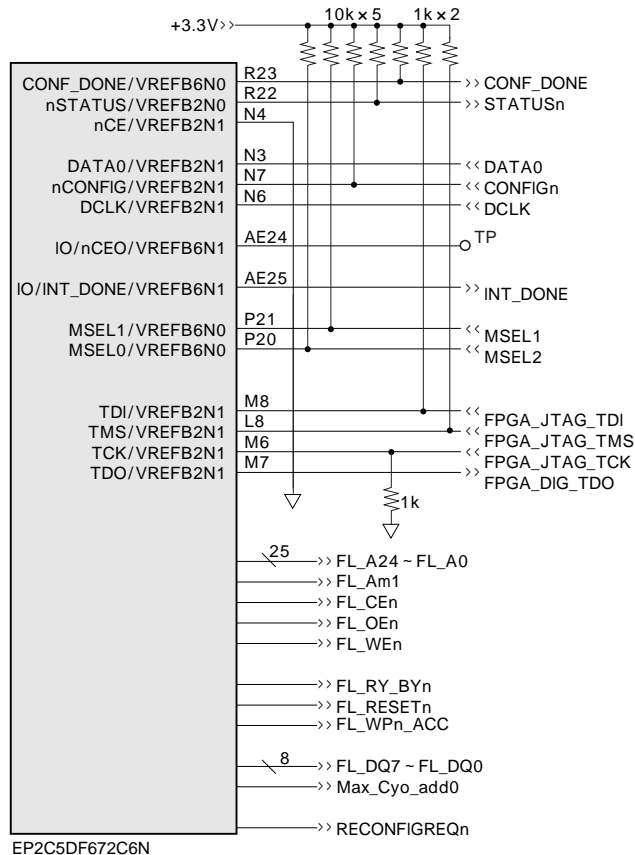
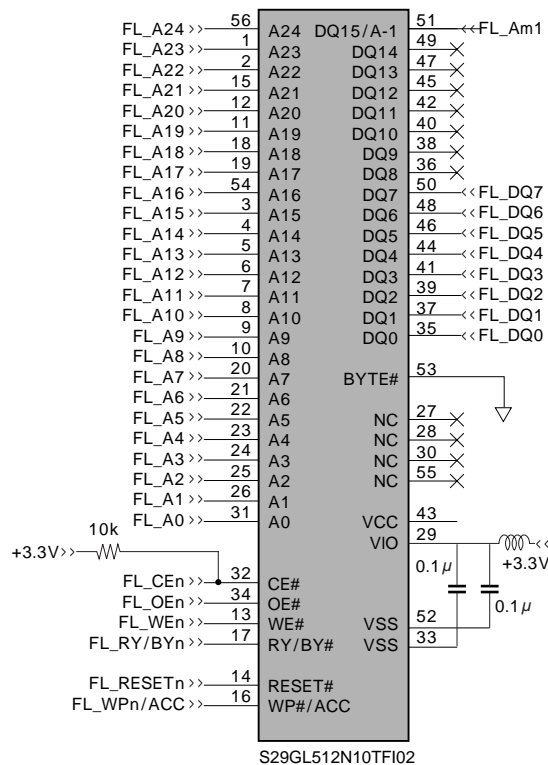
1

（a）コンフィグレーション制御用CPLD部

（b）FPGA部

（c）フラッシュ・メモリ部

図8
CFI フラッシュ・メモリ対応PS モード用コンフィグレーション制御回路

リスト1 コンフィグレーション制御回路のVerilog HDL記述

```
module monterey_maxii_ps (
  // Interface to Spantion S29GL512N11 Flash-Memory
  inout [23:0] FL_A,
  inout [7:0]  FL_DQ,
  inout        FL_Am1,
  inout        FL_CEn,
  inout        FL_OEn,
  inout        FL_WEn,
  inout        FL_RY_BYn,
  inout        FL_RESETn,
  inout        FL_WPn_ACC,
  // Interface to EP2C35F672 configration port
  input        C_conf_done,
               C_nStatus,
               C_reConfigREQn,
  output       C_FL_DQ0,
               C_nConfig,
               C_DCLK,
  input        C_init_done,
  output [1:0] C_msel,
  // interface to LED
  output       LED_R, LED_G, LED_B,
  output       LED_R_Out, LED_G_Out, LED_B_Out,
  output          SW_5_LED,mx_resetn_LED,p_sw_led,
  // for debug
  output       p_sw_state1, sw5_state1,
  // General signals
  input        sys_clk_33m,
               p_sw, sw5,
               mx_resetn
);

   parameter period = 15152, delay_5ns=5000;   // 33Mhz = 30.303ns = clock period = 15152ps * 2
   parameter led_width = 3, addr_width = 26;   // led R[2], G[1], B[0],
   parameter div_width = 4,                    // div_cnt = 1/16
             delay_40us = 1321/6;      // Configuration @5MHz wait 40us :  30.303ns * 1321 = 40030.263 ns
   parameter cyclone_ii_cd2um = 300; // conf_done to init_done clock cyclone

 `define  chat_width  10
   parameter start = 3'h0,
             wait_C_nConfig_40us = 3'h1, wait_reconfg_40us = 3'h2,
             status = 3'h3, wait_41us = 3'h4,
             config_busy = 3'h5, init = 3'h6,
             done = 3'h7;
  reg [2:0]  conf_state;
  reg [7:0]  data_reg;
  reg        data_state;
// p_sw and sw5 Chattering removal circuit
  reg [`chat_width-1:0]  sw_cnt;
  reg        p_sw_state, sw5_state;
  reg [20:0] FL_A_l ; // { FL_A[19:0], FL_Am1 };
  reg [7:0]  p_sw_shift, sw5_shift;
  reg [3:0]  div_cnt;
  reg        cnt_clr;
  reg        C_DCLK_r;
//  reg [1:0]  C_DCLK_state;
  reg [10:0] wait_cnt;
  reg        power_on_state;
  reg [1:0]  fm_reconfig_reg;
  reg        fm_reconfig_puls;
  wire       data_ready;
  reg        address_up_enb;
  reg [1:0]  sw5_st_shift;

  function [2:0] led_disp;
  input          p_sw_state;
  input  [2:0]   conf_state;
//  case ({FL_A_h, conf_state })
  casex ( {p_sw_state ,conf_state})
      {1'b0,start}                : led_disp = 'b001;
      {1'b0,wait_C_nConfig_40us}  : led_disp = 'b001;
      {1'b0,wait_reconfg_40us}    : led_disp = 'b001;
      {1'b0,status}               : led_disp = 'b001;
      {1'b0,wait_41us}            : led_disp = 'b001;
      {1'b0,config_busy}          : led_disp = 'b110;
      {1'b0,init}                 : led_disp = 'b110;
      {1'b0,done}                 : led_disp = 'b011;

      {1'b1,start}                : led_disp = 'b100;
      {1'b1,wait_C_nConfig_40us}  : led_disp = 'b100;
      {1'b1,wait_reconfg_40us}    : led_disp = 'b100;
```

1

リスト1　コンフィグレーション制御回路のVerilog HDL記述（つづき）

```
       {1'b1,status}                 : led_disp = 'b100;
       {1'b1,wait_41us}              : led_disp = 'b100;
       {1'b1,config_busy}            : led_disp = 'b011;
       {1'b1,init}                   : led_disp = 'b011;
       {1'b1,done}                   : led_disp = 'b110;
       default                       : led_disp = 'b000;
     endcase
      endfunction
     wire sw_carry   = ( sw_cnt == `chat_width'hffffff );

     wire enb_p_sw        = (  ( conf_state == done ) );
     wire conf_state_busy = (    ( conf_state == status )
                              || ( conf_state == wait_41us )
                              || ( conf_state == config_busy )
                              || ( conf_state == init ) );
     wire p_sw_pos   = ( p_sw_shift == 8'b11111111) && enb_p_sw;
     wire p_sw_neg   = ( p_sw_shift == 8'b00000000) && enb_p_sw;
     wire sw5_pos    = ( sw5_shift  == 8'b01111111) && enb_p_sw;
     wire sw5_neg    = ( sw5_shift  == 8'b10000000) && enb_p_sw;
     wire sw5_pulse  =   ( sw5_st_shift == 2'b10 );

/////////////////////////////////////////////////////////////////
//   0x000_0000 - 0x0ff_ffff  : user software FL_DQ area
//   0x100_0000 - 0x17f_ffff  : bank1 config FL_DQ area
//   0x180_0000 - 0x200_0000  : bank2 config FL_DQ area
/////////////////////////////////////////////////////////////////
     wire [24:0] FL_A_w   = ( ~p_sw_state ) ? { 4'h8, FL_A_l } : { 4'hc, FL_A_l };
     assign { FL_A, FL_Am1 }  = ( conf_state_busy ) ? FL_A_w : 'hzzzzzzzz;
     assign { LED_R, LED_G, LED_B } = led_disp ( p_sw_state ,conf_state);
     assign { LED_R_Out, LED_G_Out, LED_B_Out } = { ~LED_R, ~LED_G, ~LED_B };

     assign FL_WEn     = 'bz;
     assign FL_OEn     = (   ( conf_state == wait_41us )
                          || ( conf_state == config_busy )) ? 'b0 : 'bz;

     assign FL_WPn_ACC = 'b1;
     assign C_msel     = 'b01; // set PS configration mode
     assign FL_DQ      = 'hzz;
     assign C_FL_DQ0   = (   ( conf_state == wait_41us )
                          || ( conf_state == config_busy )
                          || ( conf_state == init ) ) ? data_reg[0] : 'bz;
     assign SW_5_LED =   sw5_state;
     assign mx_resetn_LED =   ~mx_resetn;
     assign p_sw_led =   p_sw_state;

     assign C_nConfig  = (   ( conf_state == start )
                          || ( conf_state == wait_C_nConfig_40us )
                                                        || ( conf_state == wait_reconfg_40us )) ? 'b0 : 'bz;
     assign FL_CEn     = (   ( conf_state == wait_41us )
                          || ( conf_state == config_busy )) ? 'b0 : 'bz;
     wire   C_DCLK_sig = (   ( conf_state == config_busy )
                          || ( conf_state == init )) ? C_DCLK_r : 'b0;
     assign C_DCLK     = (   ( conf_state == wait_41us )
                          || ( conf_state == config_busy )
                          || ( conf_state == init ) ) ? C_DCLK_sig : 'bz;
     assign data_ready = (   (( conf_state == config_busy ) && ( div_cnt == 'hf ))
                          || (( conf_state == wait_41us ) && ( wait_cnt == ( delay_40us - 'h01 ))));

     assign p_sw_state1 = p_sw_state;
     assign sw5_state1 = sw5_state;

always @ ( posedge sys_clk_33m or negedge mx_resetn )
  if ( ~mx_resetn )     div_cnt <= 'h0;
  else if ( conf_state == config_busy )
                        div_cnt <= div_cnt + 'h1;
       else             div_cnt <= 'h0;

always @ ( posedge sys_clk_33m or negedge mx_resetn )
  if ( ~mx_resetn )     C_DCLK_r <= 'b0;
  else if (   ( conf_state == config_busy )
           || ( conf_state == init ) )
                        C_DCLK_r <= ~C_DCLK_r;
       else             C_DCLK_r <= 'b0;

always @ ( posedge sys_clk_33m or negedge mx_resetn )
  if ( ~mx_resetn )     address_up_enb <= 'b0;
  else if ( conf_state == config_busy )
                        address_up_enb <=  ( div_cnt == 'h1 );
  else                  address_up_enb <= 'b0;

always @ ( posedge sys_clk_33m or negedge mx_resetn )
```

```
  if ( ~mx_resetn )     FL_A_l <= 'h0;
  else if ( ~cnt_clr )  FL_A_l <= 'h0;
  else if ( address_up_enb )
                        FL_A_l <= FL_A_l + 'h1;

always @ ( posedge sys_clk_33m or negedge mx_resetn )
  if ( ~mx_resetn )  begin
                        data_state <= 'b0; data_reg <= 'b0; end
  else case ( data_state )
         'b0  : if ( conf_state == wait_41us ) begin
                        data_reg <= FL_DQ; end
                  else if ( conf_state == config_busy ) begin
                        data_state <= 'b1;
                        data_reg <= FL_DQ; end
         'b1  : if ( data_ready )
                        data_reg <= FL_DQ;
                  else if ( C_DCLK_r )
                        data_reg <= { 1'b0, data_reg[7:1] };
                  else if ( conf_state == init )
                        data_state <= 'b0;
         default  : begin
                        data_state <= 'b0;
                        data_reg <= 'b0; end
        endcase

always @ ( posedge sys_clk_33m or negedge mx_resetn )
  if ( ~mx_resetn )  fm_reconfig_reg <= 'b0;
  else               fm_reconfig_reg <= ( {fm_reconfig_reg[0], C_reConfigREQn });

always @ ( posedge sys_clk_33m or negedge mx_resetn )
  if ( ~mx_resetn )  fm_reconfig_puls <= 'b0;
  else               fm_reconfig_puls <= ( fm_reconfig_reg[1:0] == 'b10 );

    always @ ( posedge sys_clk_33m or negedge mx_resetn )
        if ( ~mx_resetn ) begin
                        conf_state <= start;
                                 power_on_state <= 'b0;
                                 cnt_clr <= 'b0;
                        wait_cnt   <= 'h0; end
        else
            case ( conf_state )
                start : begin              // start state
//                      conf_state <= wait_C_nConfig_40us;
                       conf_state <= done;
//                         power_on_state <= 'b0;
                        power_on_state <= 'b1;

                                 cnt_clr <= 'b0;
                        wait_cnt   <= 'h0; end

                wait_C_nConfig_40us : begin  // wait_C_nConfig_40us state
                                           // wait C_nConfig(40us) tCFG
                                 cnt_clr <= 'b0;
                    if ( power_on_state && ( sw5_pulse || fm_reconfig_puls ) ) begin
                        conf_state <= wait_reconfg_40us;
                                        wait_cnt   <= 'h0; end
                    else if (( wait_cnt >= delay_40us ) && ~power_on_state) begin // wait 40us
                        conf_state <= status; power_on_state <= 'b1;
                        wait_cnt   <= 'h0; end
                    else begin
                        conf_state <= wait_C_nConfig_40us;
                        wait_cnt   <= wait_cnt + 'h1; end
                    end
                         wait_reconfg_40us : begin
                                        cnt_clr <= 'b0;
                    if ( wait_cnt >= delay_40us ) begin // wait 40us
                        conf_state <= status;
                        wait_cnt   <= 'h0; end
                    else begin
                        conf_state <= wait_reconfg_40us;
                        wait_cnt   <= wait_cnt + 'h1; end
                    end
                status : begin              // status state
                                            // wait C_nStatus state
                    if ( C_nStatus )
                        conf_state <= wait_41us;
                                        cnt_clr <= 'b1;
                        wait_cnt   <= 'h0; end

                wait_41us : begin           // wait_41us state
                                            // wait 41us
```

電源投入時にコンフィグレーションを始めたい場合は有効（// conf_state <= wait_C_nConfig_40us;）

電源投入時にコンフィグレーションを始めたい場合はコメント（conf_state <= done;）

電源投入時にコンフィグレーションを始めたい場合は有効（// power_on_state <= 'b0;）

電源投入時にコンフィグレーションを始めたい場合はコメント（power_on_state <= 'b1;）

1

リスト1 コンフィグレーション制御回路のVerilog HDL記述(つづき)

```
                        cnt_clr <= 'b1;
                if ( wait_cnt >= ( delay_40us )) begin
                    conf_state <= config_busy;
                                cnt_clr <= 'b1;
                    wait_cnt   <= 'h0; end
                else begin
                    conf_state <= wait_41us;
                    wait_cnt   <= wait_cnt + 'h1; end
                end

            config_busy : begin          // config_busy state
                                         // Confuguration state
                if ( ~C_nStatus )
                    conf_state <= start;
                else if ( C_conf_done )
                    conf_state <= init;
                else begin
                                cnt_clr <= 'b1; end
                end

            init : begin                 // init state
                                         // Initialize state
                wait_cnt   <= 'h0;
                if ( ~C_nStatus ) begin
                    conf_state <= start;
                                cnt_clr <= 'b0; end
                         else if ( C_init_done ) begin
                    conf_state <= done;
                                cnt_clr <= 'b0;  end
                else if ( wait_cnt >= cyclone_ii_cd2um )begin
                    conf_state <= done;
                                cnt_clr <= 'b0;  end
                else begin
                    conf_state <= init;
                    wait_cnt   <= wait_cnt + 'h1; end
                end

            done :  if ( power_on_state && ( sw5_pulse || fm_reconfig_puls ) ) begin
                        conf_state <= wait_reconfg_40us;
                                    wait_cnt   <= 'h0; end
                    else begin                 // done state
                        conf_state <= done;
                                    cnt_clr <= 'b0;  end
            default : begin
                        conf_state <= start;
                                    cnt_clr <= 'b0;  end

        endcase

// p_sw and sw5 Chattering removal circuit
  always @ ( posedge sys_clk_33m or negedge mx_resetn )
    if ( ~ mx_resetn )   sw_cnt <= `chat_width'h0;
    else if ( sw_carry ) sw_cnt <= `chat_width'h0;
         else            sw_cnt <= sw_cnt + `chat_width'h1;

  always @ ( posedge sys_clk_33m or negedge mx_resetn )
    if ( ~ mx_resetn )   p_sw_shift <= 'h0;
    else if ( sw_carry && enb_p_sw )
                         p_sw_shift <= {p_sw_shift[6:0], ~p_sw };

  always @ ( posedge sys_clk_33m or negedge mx_resetn )
    if ( ~ mx_resetn )   sw5_shift <= 'h0;
    else if ( sw_carry && enb_p_sw )
                         sw5_shift <= {sw5_shift[6:0], ~sw5 };

    reg  [1:0] p_sw_ss_state;
  always @ ( posedge sys_clk_33m or negedge mx_resetn )
    if ( ~ mx_resetn )   p_sw_ss_state <= 2'b00;
    else case ( p_sw_ss_state )
         2'b00 : if ( p_sw_pos && enb_p_sw )  p_sw_ss_state <= 2'b01;
         2'b01 : if ( p_sw_neg && enb_p_sw )  p_sw_ss_state <= 2'b10;
         2'b10 : if ( p_sw_pos && enb_p_sw )  p_sw_ss_state <= 2'b11;
         2'b11 : if ( p_sw_neg && enb_p_sw )  p_sw_ss_state <= 2'b00;
        endcase

  always @ ( posedge sys_clk_33m or negedge mx_resetn )
    if ( ~ mx_resetn )   p_sw_state <= 'b0;
    else case ( p_sw_state )
         'b0 : if ( (p_sw_ss_state == 2'b01) && enb_p_sw ) p_sw_state <= 'b1;
         'b1 : if ( (p_sw_ss_state == 2'b11) && enb_p_sw ) p_sw_state <= 'b0;
         endcase
```

```
always @ ( posedge sys_clk_33m or negedge mx_resetn )
  if ( ~ mx_resetn )   sw5_state <= 'b0;
  else case ( sw5_state )
       'b0 : if ( sw5_pos && enb_p_sw ) sw5_state <= 'b1;
       'b1 : if ( sw5_neg && enb_p_sw ) sw5_state <= 'b0;
       endcase
always @ ( posedge sys_clk_33m or negedge mx_resetn )
  if ( ~ mx_resetn )   sw5_st_shift <= 'h0;
  else if ( enb_p_sw )
                       sw5_st_shift <= {sw5_st_shift[0], sw5_state };
       else            sw5_st_shift <= 'h0;

endmodule
```

この回路では，二つのスイッチを使用します．コンフィグレーションを実行するスイッチ(MAX_resetn)と使用するコンフィグレーション・データを選択するスイッチ(MAX_P_SW)です．MAX_P_SWは，今回はコンフィグレーション・データ選択レジスタをトグルするようにしました．

スイッチとは別に，FPGAから再コンフィグレーションを要求するための信号(C_reConfigREQn)と使用するコンフィグレーション・データを選択する信号(C_reConfigSel(Max_cyc_add0))を設けています．

LEDはデバッグ用です．

リスト1では，電源投入時はコンフィグレーションを行わず，ReConfigReq_swまたはC_reConfigReqnの入力により開始する設計にしています．これは，最初はフラッシュ・メモリにコンフィグレーション・データを書き込む前にコンフィグレーションを実行しないようにするための処置です．電源投入時に自動的にコンフィグレーションを開始させたいときには，**リスト1**のコメントを参照して修正します．

FPGA開発ツールのQuartus IIでは，init_doneを使用するように設定する必要があります．

参考・引用*文献

(1) Cyclone III デバイス・ハンドブック Volume 1，Altera，2007年9月．

(2) Configuration Handbook，Altera，2007年9月．

(3) Using the Serial FlashLoader with the Quartus II Software, Application Note 370，Altera，2006年7月．

うえしま・ひでたか
(株)アルティマ

<筆者プロフィール>

上島秀隆．システムハウスでディジタル回路設計を覚え，その後CPLDを扱う半導体商社でFAEとして従事．アルティマには1994年に入社．大阪営業所所属のFAEとして現在も活躍中．最近は組み込みプロセッサのスペシャリストとして活躍するとともに東京・大阪のセミナ講師を精力的にこなす主席技師．